S+WAVELETS: ALGORITHMS AND TECHNICAL DETAILS

Andrew G. Bruce
Hong-Ye Gao
David Ragozin

StatSci Division
MathSoft, Inc.
1700 Westlake Ave. N, Suite 500
Seattle, WA 98109

Last revised
January 16, 1995

This research was supported by the Office of Naval Research (contract: N00014-92-0066, technical monitor: Gary Hewer) and NASA, Stennis Space Center (SBIR Phase II contract NAS13-587, technical monitor: Bruce Davis).

S+WAVELETS: Algorithms and Technical Details

Abstract

A complete description is given for the algorithms in S+WAVELETS software toolkit for wavelet and cosine packet analysis. These algorithms include wavelet transforms, wavelet packet transforms, cosine packet transforms, and non-decimated wavelet transforms. Implementations for the transforms and their inverses are given for a variety of boundary treatment rules, including periodic, reflection, interval wavelets (Cohen et al. [CDV93]), and zero/polynomial extension. In addition, modifications to the standard algorithms are given to handle signals or images with dimensions not divisible by a power of two.

Keywords: Discrete wavelet transform, discrete cosine transform, wavelet packet transform, boundary rules, algorithms, S+WAVELETS.

1 Introduction

This paper describes the algorithms in the S+WAVELETS toolkit for the wavelet and cosine packet analysis. S+WAVELETS is an S-PLUS module for wavelet analysis of signals and images. See Bruce and Gao [BG94, BG95] for details concerning the S+WAVELETS.

The classical one-dimensional discrete wavelet transform maps a continuous signal $f(t) \in L_2(R)$ to the wavelet coefficients $\{s_{J,n} : n \in \mathbf{Z}\}$ and $\{d_{j,n} : j \leq J, n \in \mathbf{Z}\}$. The signal is defined on the entire real line R and the transform has an infinite number of coefficients. Given coefficients $\{s_{k,n} : n \in \mathbf{Z}\}$, it is straightforward to compute the coefficients at successively coarser levels $j > k$ using the pyramid algorithm [Mal89].

Application of the discrete wavelet transform to problems involving real data involves several conceptual and computational hurdles. Typically, we are only concerned with finitely sampled signals $f_1, f_2, \cdots, f_N$. The length of the signal sequence N is not necessarily a power of two, and we may need to keep all of the data for the analysis. In this paper, we address these issues by discussing a suite of algorithms for applying the discrete wavelet transform and the wavelet packet transform to finite length signals of arbitrary length.

To handle finite signals, special treatment is required at the boundaries. In S+WAVELETS, several boundary treatment rules are available. These rules fall into one of three categories:

1. **Infinite Extension Model:** The finite signal is extended to an infinite signal $\cdots, \tilde{f}_{-1}, \tilde{f}_0, \tilde{f}_1, \cdots$ and the the classical wavelet transform is applied to the infinite sequence of coefficients. The transform coefficients are then selected from the infinite transform which correspond to the original finite signal. Infinite extension boundary rules in S+WAVELETS are periodic, reflection, zero, and polynomial extension.

2. **Recursive Extension:** An *ad hoc* boundary rule can be defined by recursively extending the signal at each filtering step in the wavelet or wavelet packet transform. Infinite extension requires either storage of extra coefficients or extremely messy bookkeeping. As a simple alternative to infinite extensive, S+WAVELETS offers zero and polynomial recursive extension rules.

3. **Wavelets on an Interval Model:** Cohen, Daubechies, and Vial [CDV93] formally defined a new wavelet transform on the compact set $[a, b]$. This transform may be different from the classical wavelet transform defined on the real line $\mathcal{R}$. The periodic and reflection infinite extension rules can also be viewed as wavelet transforms restricted to a compact interval.

Cosine packet analysis is commonly known as *local cosine* analysis, and was first introduced by Ronald Coifman and Yves Meyer [CM91]. The term "cosine packets" was coined by David Donoho, another leading wavelets researcher, because cosine packet analysis is a mirror image of wavelet packet analysis [CMQW90]. The difference is that localized cosine functions are used instead of wavelet packet functions.

The *Fourier cosine transform* (FCT) of a signal $f(t)$ is given by

$$g(\omega) = \sqrt{\frac{2}{\pi}} \int_0^\infty x(t) \cos(\omega t) dt \tag{1}$$

The discrete cosine transform is a discretized version of (1). There are four commonly used orthogonal discrete cosine transforms: DCT-I, DCT-II, DCT-III, and DCT-IV. S+WAVELETS provides functions for two of these transforms: DCT-II and DCT-IV.

Cosine packet analysis is a generalization of the discrete cosine transform (DCT). The DCT is widely used in signal processing and image processing, and is particularly valuable for coding and data compression applications. One drawback of the DCT is the abrupt "cutoff" implicit in dividing the signal into disjoint blocks. This can cause undesirable block effects, such as "Gibbs phenomena." To avoid the problems caused by the abrupt cutoff, Coifman and Meyer [CM91] introduced a new type of localized cosine transform with smooth cutoffs (tapers).

A cosine packet function is obtained by damping a cosine function down to zero on an interval I using a *taper function* or *bell function* β_I. Type II and type IV cosine packet functions with frequency k defined on the interval $I = [a, b]$ are given by

$$C_k^{II}(t) = \sqrt{\frac{2}{\Delta_I}} \beta_I(t) \cos\left(\frac{\pi k(t-a)}{\Delta_I}\right) \tag{2}$$

$$C_k^{IV}(t) = \sqrt{\frac{2}{\Delta_I}} \beta_I(t) \cos\left(\frac{\pi(k+1/2)(t-a)}{\Delta_I}\right) \tag{3}$$

where $\Delta_I = b - a$. In order to define orthogonal transforms, a very special kind of tapering function is needed.

Section 2 gives the "vanilla" algorithm for computing the discrete wavelet transform and wavelet packet transform. Treatment of the boundaries is ignored in the vanilla algorithm. Section 3 gives an overview of the boundary rules for the discrete wavelet transform and for the wavelet packet transform. Section 4 gives the wavelet convolution and down-sampling algorithms. Section 5 gives the wavelet convolution and up-sampling algorithms. Section 6 gives the algorithm for computing non-decimated (over-sampled) wavelet transform, along with an algorithm for

wavelet convolution and dilation. Section 7 describes the algorithms used to compute the discrete cosine transforms (DCT-II and DCT-IV). The Section 7.3 describes the boundary extension rules for cosine packets. Section 8 describes the algorithm for computing a cosine packet transforms. The inverse cosine packet transform is discussed in section 9. The equations for the tapers are given in Appendix section B.

This paper assumes the reader is familiar with the basic methods and algorithms for wavelets and wavelet packets. For background on wavelets, refer to [Str89, Dau92, Chu92, JLS94]. For background on wavelet packets, refer to [CW92, CMW92, Wic94]. For further discussion regarding the algorithms and methods behind cosine packet analysis (local cosine analysis), refer to the book by Wickerhauser [Wic94].

2 Wavelet and Wavelet Packet Transforms: Vanilla Algorithms

The discrete wavelet transform (DWT) and the inverse discrete wavelet transform (IDWT) use Mallat's[Mal89] remarkable fast pyramid algorithms. The wavelet packet transform (WPT) and inverse wavelet packet transform (IWPT) involve a generalization of the pyramid algorithm [CMQW90]. Mallat's wavelet pyramid algorithm has its roots in the pyramid algorithm of Burt and Adelson [BA83]. Chapter 3 of [Mey93] gives a historical perspective of the pyramid algorithm.

The forward algorithms involve convolution with low-pass and high-pass *analysis* filters followed by *down-sampling* (decimation) operator. The backwards (inverse) algorithms involve convolution with low-pass and high-pass *synthesis* filters followed by *up-sampling* (zero-padding) operator. The convolution and down-sampling operators are described in section 4, and the convolution and up-sampling operators are described in section 5. In this section, we describe the "vanilla" wavelet transform and wavelet packet transform algorithms.

2.1 Forward Algorithm to Compute the DWT

The DWT pyramid algorithm is represented by figure 1. The basic components in each stage of the pyramid are two *analysis* filters—a low-pass filter L and a high-pass filter H—and a *decimation-by-two* operation. The down-sampling (decimation) operation, indicated by a downward pointing arrow with the number 2, consists of deleting every other value of the filter outputs.

Start with a signal $\mathbf{z} = (z_1, z_2, \ldots, z_n)'$. Let $W_{F,\downarrow}$ be the convolution and down-sampling operator for filter F. The pyramid algorithm of J multiresolution levels consists of the following steps:

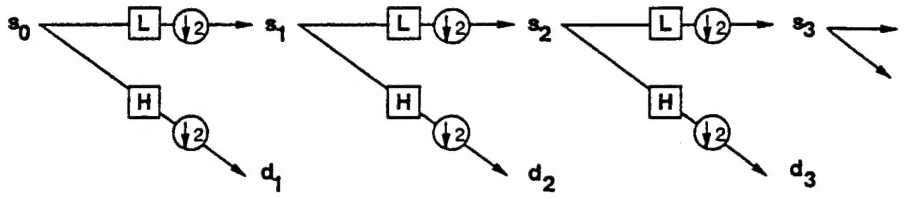

Figure 1: DWT pyramid algorithm.

[0] Initialize the input $\mathbf{s}_0 = (s_{0,1}, s_{0,2}, \ldots, s_{0,n})'$ to the pyramid algorithm as follows:

$$s_{0,i} = z_i \qquad i = 1, 2, \cdots, n.$$

Initialize the level index $j = 1$.

[1] Apply the convolution and down-sampling operator to $\mathbf{s}_{j-1}$ with the high pass filter H to obtain the level j detail coefficients

$$\mathbf{d}_j = W_{H,\downarrow}(\mathbf{s}_{j-1}).$$

Store the $\mathbf{d}_j$ coefficients.

[2] Apply the convolution and down-sampling operator to $\mathbf{s}_{j-1}$ with the low pass filter L to obtain the level j smooth coefficients

$$\mathbf{s}_j = W_{L,\downarrow}(\mathbf{s}_{j-1}).$$

If $j < J$, then increment $j = j + 1$, and go to step 1. Otherwise, store the $\mathbf{s}_j$ coefficients and quit.

The output of the algorithm is the set of DWT coefficients $\mathbf{d}_1$, $\mathbf{d}_2$, $\cdots$, $\mathbf{d}_J$, and $\mathbf{s}_J$.

2.2 Backwards Algorithm to Compute the IDWT

The inverse discrete wavelet transform (IDWT) algorithm, represented by figure 2, inverts the DWT pyramid algorithm in a straightforward manner. The basic components in each stage of the reconstruction are the *synthesis* low and high pass filters L^* and H^* and an *up-sample-by-two* operation. The up-sample operation, indicated by an upward pointing arrow with the number 2, consists of inserting zeros between every other value of the filter inputs.

Start with the DWT coefficients $\mathbf{d}_1$, $\mathbf{d}_2$, $\cdots$, $\mathbf{d}_J$, and $\mathbf{s}_J$. Let $W_{F,\uparrow}$ be the convolution and up-sampling operator for filter F. The backwards pyramid algorithm proceeds as follows:

[0] Initialize the level index $j = J$.

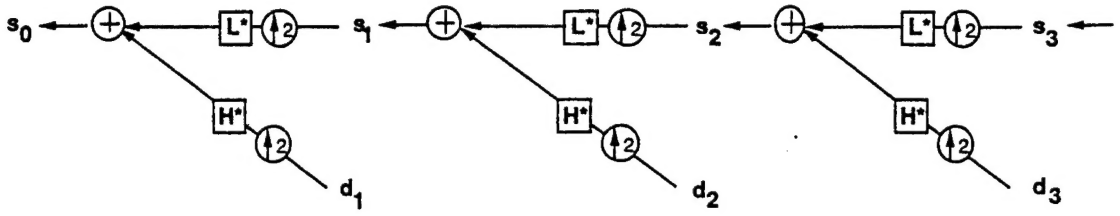

Figure 2: Reconstruction algorithm for the discrete wavelet transform.

[1] Apply the convolution and up-sampling operators to the level j coefficients to obtain the level $j-1$ smooth coefficients:

$$\mathbf{s}_{j-1} = W_{L,\uparrow}(\mathbf{s}_j) + W_{H,\uparrow}(\mathbf{d}_j).$$

[2] If $j > 1$, then go to step 1. Otherwise, for $j = 1$, set the output signal $\mathbf{z}$ to

$$z_i = s_{0,i} \qquad i = 1, 2, \cdots, n.$$

2.3 Wavelet Packet Table

The pyramid algorithm for wavelet packet table is represented by figure 3. The basic components in each stage of the pyramid, similar to DWT case, are two *analysis* filters—a low-pass filter L and a high-pass filter H—and a *decimation-by-two* operation. The down-sampling (decimation) operation, indicated by a downward

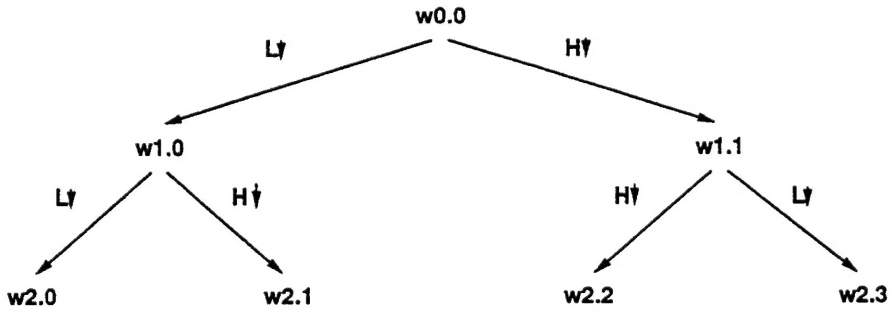

Figure 3: Pyramid algorithm for wavelet packet table.

pointing arrow with the number 2, consists of deleting every other value of the filter outputs.

Start with a signal $\mathbf{z} = (z_1, z_2, \ldots, z_n)'$. Let $W_{F,\downarrow}$ be the convolution and down-sampling operator for filter F. The pyramid algorithm of J multiresolution levels consists of the following steps:

[0] Initialize the input $\mathbf{w}_{0,0} = (w_{0,0,1}, w_{0,0,2}, \ldots, w_{0,0,n})'$ to the pyramid algorithm as follows:

$$w_{0,0,i} = z_i \qquad i = 1, 2, \cdots, n.$$

Initialize the level index $j = 1$.

[1] For each $b = 0, 1, \ldots, 2^{j-1} - 1$, apply the convolution and down-sampling operator to $\mathbf{w}_{j-1,b}$ with the high pass filter H and the low pass filter L to obtain the level j coefficients. The coefficients are stored in sequency order: if b is even,

$$\begin{aligned} \mathbf{w}_{j,2b} &= W_{L,\downarrow}(\mathbf{w}_{j-1,b}) \\ \mathbf{w}_{j,2b+1} &= W_{H,\downarrow}(\mathbf{w}_{j-1,b}); \end{aligned}$$

and if b is odd,

$$\begin{aligned} \mathbf{w}_{j,2b} &= W_{H,\downarrow}(\mathbf{w}_{j-1,b}) \\ \mathbf{w}_{j,2b+1} &= W_{L,\downarrow}(\mathbf{w}_{j-1,b}). \end{aligned}$$

[2] If $j < J$, then increment $j = j + 1$, and go to step 1. Otherwise quit.

The output of the algorithm is the set of wavelet packet coefficients $\{\mathbf{w}_{j,b}\}$, which are stored as a long vector (except when `boundary=''infinite''`) $(w_{1,0}, w_{1,1}, w_{2,0}, \ldots, w_{J,2^J-1})$.

2.4 Inverse Wavelet Packet Transform

The inverse wavelet packet transform (IWPT) algorithm inverts the wavelet packet pyramid algorithm. Start with a table of wavelet packet coefficients $\{\mathbf{w}_{j,b}\}$ where $j \in \{0, \ldots, J\}$ and $b \in \{0, \ldots, 2^j - 1\}$ (to handle matching pursuits these do not necessarily have to correspond to an orthogonal basis).

[0] Initialize a logical vector of length $2^(J + 1) - 1$ indicating whether the (j, b) block is in the wavelet packet table. Initialize the current level $j = J$.

[1] For each $b = 0, 1, \ldots, 2^{j-1} - 1$, if block (j, b) is in the table, apply the convolution and up-sampling operator using the appropriate filter. If $b\%\%4 = 0$ or $b\%\%4 = 3$, where $\%\%$ is the modulus operator, then used a low-pass filter L. Otherwise use a high-pass filter H. Add the result to block $(j - 1, \lfloor b/2 \rfloor)$ and mark that block as in the wavelet packet table.

[2] If $j > 0$, then decrement $j = j - 1$, and go to step 1. Otherwise quit.

The output of the algorithm is the reconstructed vector.

3 Overview of Boundary Rules for Wavelets

There are five basic types of boundary treatment rules for wavelet analysis supported in S+WAVELETS: periodic, reflection, recursive extension (zero and polynomial), interval, and infinite extension (zero and polynomial). These are described below.

Periodic Extension

The original series $z_1, z_2, \ldots, z_n$ is assumed to be n periodic, so $z_i = z_{i \oplus n}$. Equivalently, the wavelets are assumed to be periodic on the interval $[0, n]$.

Reflection Extension

The original series $z_1, z_2, \ldots, z_n$ is reflected at the boundaries and then periodically extended using the algorithm given by [Bri92]. The reflection boundary correction gives perfect reconstruction only for symmetric wavelets (i.e., the biorthogonal wavelets).

Recursive Extension

At each filtering step, the coefficients are padded at the beginning and the end of the signal with zeros or with a polynomial extension. Three polynomial extension rules are implemented in S+WAVELETS: poly0 (the first and last value are repeated), poly1 (the beginning and the end of the signal are extended using a polynomial of degree one fit to the first two and last two values respectively), and poly2 (the beginning and the end of the signal are extended using a polynomial of degree two fit to the first three and last three values respectively).

Interval Wavelets

This rule corresponds to the special wavelet functions at the boundaries defined by Cohen, Daubechies and Vial[CDV93]. The boundary wavelets are zero outside of the range of the data. The transform retains the orthogonality properties of the "classical" wavelet transform and is numerically stable.

For the interval wavelet, then a preconditioning transform can be (optionally) applied to the signal before applying the interval wavelet transform. The preconditioning transform preserves the "vanishing moment" property of wavelets at the expense of introducing an additional non-orthogonal transform. See [CDV93] for details.

Infinite Extension

The original series $z_1, z_2, \ldots, z_n$ is extended once at the beginning of the filtering procedure using a zero or polynomial extension. This is similar to the zero and polynomial rules above, for which the coefficients are extended at each filter step. For the polynomial extension, a polynomial of degree pdeg is fit using a fraction pfrac of the data ($0 \leq$ pfrac < 1).

The infinite boundary rule produces an infinite set of wavelet coefficients. However, only a finite number of coefficients is stored. Since a polynomial extension rule is used, the stored coefficients can be used to compute the remaining coefficients.

Unlike the other boundary rules, the infinite boundary treatment is an "expansionist" transform. This means that you end up with more coefficients than original sample values. For a series of length n, you obtain $n + p$ wavelet coefficients where $0 < p \ll n$, independent of n. As a result, the output data structure for a transform computed with the infinite boundary rule is different than transforms computed with other rules (the transform is a "crystal list" object rather than a "crystal vector" object: see [BG95]).

4 Convolution and Downsampling Algorithms

Let $X = (x_1, \ldots, x_n)$ be the input signal and let $f = (f_1, \ldots, f_m)$ be the filter. The "generic" convolution and down-sampling operation is given by

$$y_k = \sum_{i=1}^{m} f_i x_{2k+i-2} \tag{4}$$

Since x_i is only defined for $i = 1, \ldots, n$, the summation in (4) needs to be modified at the beginning and end of the signal. The way in which the summation is handled at the boundaries, as well as the length of the output signal, depends on the boundary rule. This section gives the algorithms for the convolution and down-sampling operator (4) for the boundary rules of section 3.

Define the operator "convdown.general" by (4) with k restricted to values for which the filter does not extend beyond the ends of the input signal

$$k = 1, 2, \ldots, \left\lfloor \frac{n - m + 2}{2} \right\rfloor.$$

The length of the output signal of the convdown.general operator, $\ell = \left\lfloor \frac{n-m+2}{2} \right\rfloor$, is in general smaller than the desired length. In order to obtain enough output coefficients ($\lfloor n/2 \rfloor$ or $\lfloor n/2 \rfloor + 1$), extra values (roughly $m - 2$) must be padded to X before calling the convdown.general operator.

The extra values can be all padded at the begining or at the end of X, or partly at the begining and partly at the end. In S+WAVELETS, if the number of padding

values is even, an equal number of values are padded at the beginning and end. If the number of padding values is odd, then one more value is padded at the end.

In the algorithms below (except for the reflection boundary rule) if f is of odd length, we pad a zero to f to make it even length. The filter padding rule is as follows: if f is a low-pass filter, $\tilde{f} = (f, 0)$; if f is a high-pass filter, $\tilde{f} = (0, f)$. For the reflection case, special filtering rules apply to odd length filters.

4.1 Periodic

The input signal X is assumed to be periodic with period n. The algorithm consists of the following two steps:

[1] extend the input signal

$$\tilde{X} = (x_{n-p+1}, \ldots, x_n,\ x_1, \ldots, x_n,\ x_1, \ldots, x_p).$$

[2] apply the general convdown operator:

$$Y = \text{convdown.general}(\tilde{X},\ f).$$

Note: If X is assumed to be periodic, then Y is also periodic. When n is even, the period of Y is $n/2$. When n is odd, the period of Y is still n. Hence, sample size of the original signal is restricted to multiples of 2^J for the periodic boundary rule.

4.2 Reflection

The input signal X is reflected and then (implicitly) periodized using the approach described by [Bri92]. When a symmetric/antisymmetric filter is applied, the filtered vector Y is of the same reflection/periodicity property. Following the notation of [Bri92], we use the "$E(1,1)$ extension" for odd length filter (i.e. m is odd) and the "$E(2,2)$ extension" for even length filter. The algorithm consists of the following two steps:

[1] extend the input signal X to get $\tilde{X}$:

- n is even and m is even

$$\tilde{X} = (x_p, \ldots, x_1,\ x_1, \ldots, x_n,\ x_n, \ldots, x_{n-p+1}).$$

- n is odd and m is even, if f is a low-pass filter

$$\tilde{X} = (x_p, \ldots, x_1,\ x_1, \ldots, x_n,\ x_n, \ldots, x_{n-p})$$

and if f is a high-pass filter

$$\tilde{X} = (x_p, \ldots, x_1,\ x_1, \ldots, x_n,\ x_n, \ldots, x_{n-p+1}).$$

- n is even and m is odd, if f is a low-pass filter

$$\tilde{X} = (x_{p+2}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p});$$

if f is a high-pass filter

$$\tilde{X} = (x_{p+1}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p-1});$$

if f is a low-pass dual filter

$$\tilde{X} = (x_{p+1}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p-1});$$

and if f is a high-pass dual filter

$$\tilde{X} = (x_{p+2}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p}).$$

- n is odd and m is odd, if f is a low-pass filter

$$\tilde{X} = (x_{p+2}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p-1});$$

if f is a high-pass filter

$$\tilde{X} = (x_{p+1}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p});$$

if f is a low-pass dual filter

$$\tilde{X} = (x_{p+1}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p});$$

and if f is a high-pass dual filter

$$\tilde{X} = (x_{p+2}, \ldots, x_2,\ x_1, \ldots, x_n,\ x_{n-1}, \ldots, x_{n-p-1}).$$

[2] apply the general convdown operator:

$$Y = \text{convdown.general}(\tilde{X},\ f).$$

4.3 Infinite (polynomial/zero)

The original signal Z is extended once at the beginning of the filtering procedure using a zero or polynomial of degree `pdeg` extension. The polynomials are fit using a fraction `pfrac` of the data.

The infinite boundary rule produces an infinite set of wavelet coefficients. Only a finite number of coefficients are actually stored. The coefficients which are not stored can be computed from the stored coefficients using a zero or polynomial extension.

Unlike the other boundary rules, the infinite boundary treatment is an "expansionist" transform. This means that in addition to the desired $\lfloor n/2 \rfloor$ interior coefficients Y, we also save extra coefficients, Y_ℓ (left boundary coefficients) and Y_r (right boundary coefficients). The number of extra coefficients depends only on M, the maximum length of analysis filter and synthesis filter. For orthogonal wavelets, the synthesis filters are the same as the analysis filters, so $M = m$.

Note: In S+WAVELETS, to handle the boundary coefficients, the output data structure (object of class `crystal.list`) is different from other boundary rules (object of class `crystal.vector`).

Let $p = m/2-1$ and $P = M/2-1$. Define $d =$ `pdeg` for the polynomial extension and $d = -1$ for the zero extension. The algorithm contains the following steps:

[1a] Initial polynomial extension: if the input signal is the original data, proceed with this step followed by step 2. Otherwise, if the input signal are the output of a previous convolution and down-sampling operations, skip to step 1b.

Let $q = 2(d + P + 1) + p$ and $Q = \max(d + 1,\ \lfloor n \times \texttt{pfrac} + 1 \rfloor)$.

- If n is even, then

$$\tilde{X} = (a_1, \ldots, a_q,\ x_1, \ldots, x_n,\ b_1, \ldots, b_q).$$

- Otherwise, if n is odd, then

$$\tilde{X} = (a_1, \ldots, a_q,\ x_1, \ldots, x_n,\ b_1, \ldots, b_{q+1})$$

The coefficients $(a_1, \ldots, a_q)$ are the polynomial or zero extrapolations of $(x_1, \ldots, x_Q)$ and the coefficients $(b_1, \ldots, b_q)$ are the polynomial or zero extrapolations of $(x_{n-Q+1}, \ldots, x_n)$. See appendix A for details on the polynomial and zero extrapolations.

Skip to step 2.

[1b] Extend X using the existing boundary coefficients (from previous filtering procedure) X_ℓ and X_r:

$$\tilde{X} = (X_\ell,\ X,\ X_r)$$

Let N be the length of $\tilde{X}$ and let $q = d + P + 1 + p$. Extend $\tilde{X}$ to obtain $\hat{X}$ to obtain

- If N is even, then

$$\hat{X} = (a_1, \ldots, a_q,\ \tilde{x}_1, \ldots, \tilde{x}_N,\ b_1, \ldots, b_q).$$

- Otherwise, if N is odd, then

$$\hat{X} = (a_1, \ldots, a_q,\ \tilde{x}_1, \ldots, \tilde{x}_N,\ b_1, \ldots, b_{q+1}).$$

The coefficients $(a_1, \ldots, a_q)$ are the polynomial or zero extrapolations of $(x_1, \ldots, x_{d+1})$ and the coefficients $(b_1, \ldots, b_q)$ are the polynomial or zero extrapolations of $(x_{N-d}, \ldots, x_N)$ (see appendix A). Note that `pfrac` is not used for this extrapolation.

[2] apply the general convdown operator to $\tilde{X}$

$$\tilde{Y} = \text{convdown.general}(\tilde{X},\ f).$$

[3] Let $b = d + P + 1$ and $n' = \lfloor (n+1)/2 \rfloor$. Obtain the interior coefficients Y, the left boundary coefficients Y_ℓ, and the right boundary coefficients Y_r by

$$\begin{aligned} Y &= (\tilde{y}_{b+1}, \ldots, \tilde{y}_{b+n'}) \\ Y_\ell &= (\tilde{y}_1, \ldots, \tilde{y}_b) \\ Y_r &= (\tilde{y}_{n'+b+1}, \ldots, \tilde{y}_{n'+2b}) \end{aligned}$$

On each end, $d + P - 1$ boundary coefficients are saved.

4.4 Interval

An alternative to the infinite padding model is to define a slightly modified wavelet transform adapted for finite length signals. [CDV93] show how to construct orthornormal wavelet bases for intervals with compact support. In the interior of the interval, the basis functions are identical to usual wavelet basis functions. However, the basis functions at the boundaries are given by specialized edge functions $\phi^0_{j,k}$ and $\psi^0_{j,k}$

The interval wavelets are implemented using special filters at the ends of the series corresponding to the truncated and orthongalized wavelet basis functions. They are only implemented for the discrete wavelet transform with "symmlets": `s4, s6, s8, s10, s12, s14, s16`. The special filters are stored as $p+1$ by $3p+2$ boundary matrices B_ℓ and B_r. Sample size n must divisible by 2^J where J is the maximum down-sampling level in the transform.

The algorithm consists of the following steps:

[0] precondition X to preserve the "vanishing moment" property of the original wavelets. This step is optional and applies only to the original series. Let P_ℓ and P_r be the left and right precondition matrices (transpose of S+WAVELETS functions `left.precondition` and `right.precondition` respectively), then $(x_1, \ldots, x_{m/2})' = P_\ell(x_1, \ldots, x_{m/2})'$ and $(x_{n-m/2+1}, \ldots, x_n)' = P_\ell(x_{n-m/2+1}, \ldots, x_n)'$.

[1] extend X by zeros:

$$\tilde{X} = (\overbrace{0,\ldots,0}^{p},\ x_1,\ldots,x_n,\ \overbrace{0,\ldots,0}^{p}).$$

[2] apply the general convdown operator:

$$Y = \text{convdown.general}(\tilde{X},\ f)$$

[3] apply the special filters to correct the first $p+1$ and last $p+1$ values of Y based on the first $3p+2$ and last $3p+2$ values of X. Let B_ℓ and B_r (S+WAVELETS functions `left.interval` and `right.interval` respectively) be the left and right boundary matrices respectively, then

$$\begin{pmatrix} y_1 \\ \vdots \\ y_{p+1} \end{pmatrix} = B_\ell \begin{pmatrix} x_1 \\ \vdots \\ x_{3p+2} \end{pmatrix}$$

$$\begin{pmatrix} y_{n/2-p+1} \\ \vdots \\ y_{n/2} \end{pmatrix} = B_r \begin{pmatrix} x_{n-3p-1} \\ \vdots \\ x_n \end{pmatrix}.$$

See [CDV93] for information on obtaining the boundary filters electronically.

4.5 Zero Recursive Extension

At each filtering step, the coefficients are padded at the beginning and the end of the signal with zeros. This boundary rule applies to arbitrary sample size and filter and is implemented as a special case of `polynomial` with $pdeg = -1$. The algorithm consists of the following two steps:

[1] extend the input signal:

- n is odd and f is a low-pass filter

$$\tilde{X} = (\overbrace{0,\ldots,0}^{p},\ x_1,\ldots,x_n,\ \overbrace{0,\ldots,0}^{p+1})$$

- otherwise

$$\tilde{X} = (\overbrace{0,\ldots,0}^{p},\ x_1,\ldots,x_n,\ \overbrace{0,\ldots,0}^{p}).$$

[2] apply the general convdown operator:

$$Y = \text{convdown.general}(\tilde{X},\ f)$$

4.6 Polynomial Recursive Extension

At each filtering step, the coefficients are padded at the beginning and the end of the signal using a polynomial extension of degree $d = \texttt{pdeg}$ fit to the first $d+1$ and last $d+1$ values respectively. This boundary rule applies to arbitrary sample size and filter. The algorithm consists of the following two steps:

[1] extend the input signal:

- n is odd and f is a low-pass filter

$$\tilde{X} = (a_1, \ldots, a_p,\ x_1, \ldots, x_n,\ b_1, \ldots, b_{p+1})$$

- otherwise

$$\tilde{X} = (a_1, \ldots, a_p,\ x_1, \ldots, x_n,\ b_1, \ldots, b_p).$$

where $(a_1, \ldots, a_p)$ are the polynomial extrapolations of $(x_1, \ldots, x_{d+1})$, and $(b_1, \ldots, b_p)$ are the polynomial extrapolations of $(x_{n-d}, \ldots, x_n)$.

[2] apply the general convdown operator:

$$Y = \text{convdown.general}(\tilde{X},\ f)$$

5 Convolution and Upsampling Algorithms

Let $Y = (y_1, \ldots, y_n)$ be the input signal and let $f = (f_1, \ldots, f_m)$ be the filter. The "generic" convolution and up-sampling operation is given by

$$x_{2k-1} = \sum_{i=1}^{m} f_{2i-1} y_{i+k-1} \tag{5}$$

$$x_{2k} = \sum_{i=1}^{m} f_{2i} y_{i+k-1} \tag{6}$$

Like the convolution and down-sampling operator, the summation in (5) and (6) needs to be modified at the boundaries.

Define the "convup.general" operator by (5) and (6) with the range of summation restricted as follows:

$$k = 1, 2, \ldots, n - p$$

where $p = m/2 - 1$. The filtered vector is of length $2(n-p)$, which is in general smaller than the desired length $2n$ or $2n-1$. Therefore, in order to recover the original vector, extra values should be padded to Y before calling convup.general operator. The extra values are padded to the beginning and end in the same manner as for the convolution and down-sampling operator.

As with the down-sampling algorithms with the exception of the reflection boundary rule, we pad a zero to f if f is of odd length. The filter padding rule is as follows: if f is a low-pass filter, $\tilde{f} = (f, 0)$; if f is a high-pass filter, $\tilde{f} = (0, f)$. For the reflection case, special filtering rules apply to odd length filters.

5.1 Periodic

When the original series X is assumed to be periodic with period $2n$, Y is also periodic with period n. Let $p = m/2 - 1$ and the algorithm consists of the following three steps:

[1] extend Y:

$$\tilde{Y} = (y_{n-p+1}, \ldots, y_n,\ y_1, \ldots, y_n,\ y_1, \ldots, y_p)$$

[2] apply the general convup operator:

$$\tilde{X} = \text{convup.general}(\tilde{Y},\ f)$$

[3] select X from $\tilde{X}$:

$$X = (\tilde{x}_{p+1}, \ldots, \tilde{x}_{p+2n}).$$

5.2 Reflection

When the filter is symmetric/antisymmetric, Y is of the same reflection/periodicity property as the original vector.

Let N be the length of the original vector, $p = \lfloor m/2 \rfloor - 1$ and $q = \lfloor (m+1)/4 \rfloor$, the algorithm consists of the following three steps:

[1] extend Y as follows:

- If N is even, m is even, then if f is low-pass filter

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_n, \ldots, y_{n-q+1})$$

else if f is a high-pass filter

$$\tilde{Y} = (-y_q, \ldots, -y_1,\ y_1, \ldots, y_n,\ -y_n, \ldots, -y_{n-q+1}).$$

- Else if N is odd and m is even, then if f is a low-pass filter

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_{n-1}, \ldots, y_{n-q})$$

else if f is a high-pass filter

$$\tilde{Y} = (-y_q, \ldots, -y_1,\ y_1, \ldots, y_n,\ 0, -y_n, \ldots, -y_{n-q+1}).$$

- Else if N is even and m is odd, then if f is a low-pass filter (not dual)

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_{n-1}, \ldots, y_{n-p+q-2});$$

else if f is a high-pass filter (not dual)

$$\tilde{Y} = (y_{q+1}, \ldots, y_2,\ y_1, \ldots, y_n,\ y_n, \ldots, y_{n-p+q-1});$$

else if f is a low-pass dual filter

$$\tilde{Y} = (y_{q+1}, \ldots, y_2,\ y_1, \ldots, y_n,\ y_n, \ldots, y_{n-p+q-1});$$

else if f is a high-pass dual filter (not dual)

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_{n-1}, \ldots, y_{n-p+q-2}).$$

- Else if N is odd and m is odd, then if f is a low-pass filter

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_n, \ldots, y_{n-p+q-1});$$

else if f is a high-pass filter

$$\tilde{Y} = (y_{q+1}, \ldots, y_2,\ y_1, \ldots, y_n,\ y_{n-1}, \ldots, y_{n-p+q-1});$$

else if f is a low-pass dual filter

$$\tilde{Y} = (y_{q+1}, \ldots, y_2,\ y_1, \ldots, y_n,\ y_{n-1}, \ldots, y_{n-p+q-1});$$

else if f is a high-pass dual filter

$$\tilde{Y} = (y_q, \ldots, y_1,\ y_1, \ldots, y_n,\ y_n, \ldots, y_{n-p+q-1}).$$

[2] apply the general convup operator

$$\tilde{X} = \text{convup.general}(\tilde{Y},\ \tilde{f})$$

where $\tilde{f} = (f, 0)$ if m is odd, and f otherwise.

[3] select X from $\tilde{X}$:

- if m is multiple of 4

$$X = (\tilde{x}_2, \ldots, \tilde{x}_{N+1}).$$

- otherwise

$$X = (\tilde{x}_1, \ldots, \tilde{x}_N).$$

5.3 Infinite (polynomial/zero)

Let N be the length of the original vector, $p = m/2 - 1$ and $P = M/2 - 1$, where M is the maximum length of analysis filter and synthesis filter. The algorithm consists of the following steps:

[1] retract the boundary coefficients Y_ℓ and Y_r,

$$Y = (Y_\ell,\ Y,\ Y_r)$$

and apply the usual polynomial extrapolation scheme to Y, let $q_1 = \left\lfloor \frac{p-\min(P,p)}{2} \right\rfloor$ and $q_2 = \left\lfloor \frac{p-\min(P,p)+1}{2} \right\rfloor$,

$$\tilde{Y} = (a_1, \ldots, a_{q_1},\ y_1, \ldots, y_N,\ b_1, \ldots, b_{q_2})$$

where $(a_1, \ldots, a_{q_1})$ are the polynomial extrapolations of $(y_1, \ldots, y_{d+1})$ and $(b_1, \ldots, b_{q_2})$ are the polynomial extrapolations of $(y_{n-d}, \ldots, y_n)$. Note that when $d = -1$, zeros are padded and when $p = P$, $q_1 = q_2 = 0$, i.e. no padding is needed.

[2] apply the general convup operator,

$$\tilde{X} = \text{convup.general}(\tilde{Y},\ f)$$

[3] get the interior, the left boundary and the right boundary coefficients: let $b = 2(d + P + 1) - \min(P, p)$,

$$\begin{aligned} X &= (\tilde{x}_{b+1}, \ldots, \tilde{x}_{b+N}) \\ X_\ell &= (\tilde{x}_{b-d-P}, \ldots, \tilde{x}_b) \\ X_r &= (\tilde{x}_{N+b+1}, \ldots, \tilde{x}_{N+b+d+P+1}) \end{aligned}$$

As in convdown case, we save $d + P - 1$ boundary coefficients on each end.

5.4 Interval

Let $q = 3p + 2$. The algorithm consists of the following steps:

[1] apply the general convup operator to Y,

$$\tilde{X} = \text{convup.general}(Y,\ f).$$

[2] extend $\tilde{X}$ to the desired length $(2n)$:

$$X = (\overbrace{0, \ldots, 0}^{p},\ \tilde{X},\ \overbrace{0, \ldots, 0}^{p}).$$

[3] apply the special filters to correct the first q and the last q values of X based on the first q and the last q values of Y. Let B_ℓ and B_r (computed from `dwt.matrix`) be the left and right boundary matrices respectively, then

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{3p+2} \end{pmatrix} = B_\ell \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix}$$
$$\begin{pmatrix} x_{2n-q+1} \\ \vdots \\ x_{2n} \end{pmatrix} = B_r \begin{pmatrix} y_{n-q+1} \\ \vdots \\ y_n \end{pmatrix}$$

The boundary matrices can be computed in the following way: Let B be the transpose of the $2q$ by $2q$ `dwt.matrix` and

$$B = \left(\begin{array}{c|c} B_\ell(L) & B_\ell(H) \\ \hline B_r(L) & B_r(H) \end{array}\right)$$

then $B_\ell = B_\ell(L)$ and $B_r = B_r(L)$ if f if a low-pass filter; and $B_\ell = B_\ell(H)$ and $B_r = B_r(H)$ if f if a high-pass filter.

[4] inverse precondition (for final step of the synthesis only).

5.5 Polynomial/Zero

Let N be the length of the original series X and $q = 2\max(M/2,\ d+1,\ 1)$ where M is the maximum length of analysis filter and synthesis filter. The algorithm consists of the following 4 steps:

[1] apply the general convup operator to Y:

$$\tilde{X} = \text{convup.general}(Y,\ f)$$

and let n be the length of $\tilde{X}$.

[2] extend $\tilde{X}$ to the desired length N,

- N is even

$$X = (\overbrace{0,\ldots,0}^{p},\ \tilde{x}_1,\ldots,\tilde{x}_n,\ \overbrace{0,\ldots,0}^{p})$$

- N is odd and f is a low-pass filter

$$X = (\overbrace{0,\ldots,0}^{p},\ \tilde{x}_1,\ldots,\tilde{x}_{n-2},\ \overbrace{0,\ldots,0}^{p+1})$$

and if f is a high-pass filter

$$X = (\overbrace{0,\ldots,0}^{p},\ \tilde{x}_1,\ldots,\tilde{x}_n,\ \overbrace{0,\ldots,0}^{p+1})$$

[3] compute the inverse `dwt.matrix` B for the dual filter. The boundary matrices are the four corner matrices of B and are different for odd N and even N (see below). Therefore there are totally 8 bounadry matrices, and they are pre-computed and stored in the `dictionary`.

[4] apply the left boundary matrix B_ℓ and the right boundary matrix B_r to correct the boundary coefficients.

- When N is even, B is a $2q$ by $2q$ matrix. If f is a low-pass filter, the boundary matrices B_ℓ and B_r are q by q matrices, and defined by

$$\begin{aligned} B_\ell &= B_{[1:q,\,1:q]} \\ B_r &= B_{[(q+1):2q,\,1:q]}; \end{aligned}$$

and if f is a high-pass filter,

$$\begin{aligned} B_\ell &= B_{[1:q,\,(q+1):2q]} \\ B_r &= B_{[(q+1):2q,\,(q+1):2q]}. \end{aligned}$$

And then

$$\begin{pmatrix} x_1 \\ \vdots \\ x_q \end{pmatrix} = B_\ell \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix}$$

$$\begin{pmatrix} x_{N-q+1} \\ \vdots \\ x_N \end{pmatrix} = B_r \begin{pmatrix} y_{N-q+1} \\ \vdots \\ y_N \end{pmatrix}.$$

- When N is odd, B is a $2q+1$ by $2q+1$ matrix. If f is a low-pass filter, the boundary matrices B_ℓ and B_r are $q+1$ by $q+1$ matrices, and defined by

$$\begin{aligned} B_\ell &= B_{[1:(q+1),\,1:(q+1)]} \\ B_r &= B_{[(q+1):(2q+1),\,1:(q+1)]}; \end{aligned}$$

and then

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{q+1} \end{pmatrix} = B_\ell \begin{pmatrix} y_1 \\ \vdots \\ y_{q+1} \end{pmatrix}$$

$$\begin{pmatrix} x_{N-q} \\ \vdots \\ x_N \end{pmatrix} = B_r \begin{pmatrix} y_{N-q} \\ \vdots \\ y_N \end{pmatrix};$$

if f is a high-pass filter, B_ℓ and B_r are $q+1$ by q matrices, and defined by

$$\begin{aligned} B_\ell &= B_{[1:(q+1),\,(q+2):(2q+1)]} \\ B_r &= B_{[(q+1):(2q+1),\,(q+2):(2q+1)]}, \end{aligned}$$

and then

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{q+1} \end{pmatrix} = B_\ell \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix}$$

$$\begin{pmatrix} x_{N-q} \\ \vdots \\ x_N \end{pmatrix} = B_r \begin{pmatrix} y_{N-q+1} \\ \vdots \\ y_N \end{pmatrix}.$$

6 Algorithms for the Non-Decimated Wavelet Transform

Let $X = (x_1, \ldots, x_n)$ be the input signal and let $f = (f_1, \ldots, f_m)$ be the filter. The "generic" convolution operation is given by

$$y_k = \sum_{i=1}^{m} f_i x_{k+i-1} \tag{7}$$

Since x_i is only defined for $i = 1, \ldots, n$, the index k of the output signal Y is restricted to $k = 1, 2, \ldots, n-m+1$. Therefore, the output signal is always shorter than the input sugnal X. In order to obtain enough output coefficients, $m-1$ extra values should be added to X. Currently, we have only implemented *periodic* boundary extension.

6.1 Non-Decimated Wavelets: Vanilla Algorithms

Define the convolution and dilation operator "convdil.general" by dilating filter f and the calling (7). For given input signal $X = (x_1, \ldots, x_n)$, filter $f = (f_1, \ldots, f_m)$ and a desired level j (therefore the dilation factor is 2^j), define the dilated filter $f^{(j)}$ by inserting $2^j - 1$ zeros in each of the adjacent f_i's, i.e.

$$f^{(j)} = (f_1, \overbrace{0, \ldots, 0}^{2^j-1}, f_2, \ldots, \overbrace{0, \ldots, 0}^{2^j-1}, f_m)$$

and then apply the "generic" convolution operator (7) to X and $f^{(j)}$.

Let m_j be the length of $f^{(j)}$, then $m_j = (m-1)(2^j - 1) + m$. Hence, for given sample size n and filter length m, the dilation level j is restricted to $j \leq \log_2(\frac{n-1}{m-1})$.

6.2 Periodic Convolution and Dilation Operator

Forward: The algorithm consists of the following steps:

[0] set level $j = 0$.

[1] at level j, let $p = \left\lfloor \frac{m_j - 1}{2} \right\rfloor$, $q = \left\lfloor \frac{m_j}{2} \right\rfloor$ and extend the input signal X

$$\tilde{X} = (x_{n-p+1}, \ldots, x_n,\ x_1, \ldots, x_n,\ x_1, \ldots, x_q).$$

[2] apply the general convolution operator (7):

$$Y = \text{convdil.general}(\tilde{X},\ \text{rev}(f),\ j).$$

[3] set $j = j + 1$. If $j < J$, go to step [1].

Inverse: The algorithm consists of the following steps:

[0] set level $j = J$.

[1] at level j, let $p = \left\lfloor \frac{m_j}{2} \right\rfloor$, $q = \left\lfloor \frac{m_j - 1}{2} \right\rfloor$ and extend the input signal X

$$\tilde{X} = (x_{n-p+1}, \ldots, x_n,\ x_1, \ldots, x_n,\ x_1, \ldots, x_q).$$

[2] apply the general convolution operator (7):

$$Y = \text{convdil.general}(\tilde{X},\ f,\ j-1)/2.$$

[3] set $j = j - 1$. If $j > 0$, go to step [1].

6.3 The À Trous Wavelet Transform

An alternative non-decimated wavelet transform is given by the "*à trous*" ("hole") algorithm: see [Dut87, She92, SMB94]. Like the non-decimated wavelet transform computed using `nd.dwt`, the *à trous* algorithm produces n wavelet coefficients at each multiresolution level for a signal with n sample values. The main difference is that the detail coefficients in the *à trous* algorithm are computed through simple differences between the smooth coefficients at different levels:

$$d_{j,k} = s_{j-1,k} - s_{j,k} \tag{8}$$

The detail coefficients produced by the `nd.dwt` function are computed using a dilated high-pass wavelet filter.

7 Computing the DCT

This section gives discusses the algorithms used to compute the DCT-II and DCT-IV transforms. For details concerning algorithms for the DCT, refer to the book by Rao and Yip [RY90].

For a discrete signal $f_1, f_2, \ldots, f_n$, the DCT-II is defined as

$$g_k^{II} = \sqrt{\frac{2}{n}} s_k \sum_{i=1}^{n} f_i \cos\left(\frac{(2i-1)k\pi}{2n}\right) \quad k = 0, 1, \ldots, n-1. \tag{9}$$

The scaling factor s_k is defined by

$$s_k = \begin{cases} 1 & \text{if } k \neq 0 \text{ and } n \\ \frac{1}{\sqrt{2}} & \text{if } k = 0 \text{ or } n \end{cases} \tag{10}$$

The inverse DCT-II is given by

$$f_i = \sqrt{\frac{2}{n}} \sum_{k=0}^{n-1} g_k^{II} s_k \cos\left(\frac{(2i-1)k\pi}{2n}\right) \quad i = 1, \ldots, n. \tag{11}$$

The DCT-IV is defined as

$$g_k^{IV} = \sqrt{\frac{2}{n}} \sum_{i=1}^{n} f_i \cos\left(\frac{(2i-1)(2k+1)\pi}{4n}\right) \quad k = 0, 1, \ldots, n-1. \tag{12}$$

The inverse DCT-IV is given by

$$f_i = \sqrt{\frac{2}{n}} \sum_{k=0}^{n-1} g_k^{IV} \cos\left(\frac{(2i-1)(2k+1)\pi}{4n}\right) \quad i = 1, \ldots, n. \tag{13}$$

For a discrete signal $\mathbf{f} = (f_1, f_2, \ldots, f_n)$, the DCT's and their inverses can be computed from the FFT of the extended signal obtained by padding p zeros at the end. Define the extension operator $E_p(\mathbf{x})$ for the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ by

$$E_p(\mathbf{x}) = \begin{cases} x_t & t = 1, 2, \cdots, n \\ 0 & t = n+1, n+2, \cdots, n+p \end{cases}$$

Define the FFT $G(\mathbf{y})$ of a vector $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ by

$$G(\mathbf{y})_k = \sum_{t=1}^{m} y_t \exp\left(-i2\pi k(t-1)/m\right) \qquad k = 0, 1, \cdots, m-1 \tag{14}$$

7.1 Algorithm for the DCT-II

For the DCT-II, pad $p = n$ zeros and compute the FFT of the signal and take the first n frequencies of $\widehat{f} = G(E_n(\mathrm{f}))$. The DCT-II coefficients $\mathbf{c} = (c_0, c_2, \ldots, c_{n-1})$, are given by

$$c_k = \begin{cases} \sqrt{\frac{2}{n}}\left(\cos(\frac{k\pi}{2n})\mathrm{Re}(\widehat{\mathrm{f}}_\mathrm{k}) + \sin(\frac{k\pi}{2n})\mathrm{Im}(\widehat{\mathrm{f}}_\mathrm{k})\right) & k \neq 0 \\ \sqrt{\frac{1}{n}}\widehat{f}_0 & k = 0 \end{cases} . \tag{15}$$

The inverse DCT-II can be obtained by padding $p = 3n$ zeros to the DCT-II coefficients $\mathbf{c}$, dividing the first DCT coefficient c_0 by $\sqrt{2}$, and taking the FFT. From the FFT $\widehat{c}_k$, the original signal is reconstructed as follows:

$$f_k = \sqrt{\frac{2}{n}}\mathrm{Re}(\widehat{\mathrm{c}}_{2(\mathrm{k}-1)}) \qquad \mathrm{k} = 1, 2, ..., \mathrm{n}. \tag{16}$$

For signals of length 2^J, the DCT-II is obtained from the discrete Hartley transform (DHT). This is a fast algorithm which avoids the need to double the signal length (as is the case when the DCT-II is computed from the FFT) [Mal86]. The discrete Hartley transform [SJBH85] is defined by

$$H_k = \sum_{t=1}^{n}\left(\cos\left(\frac{2k(t-1)\pi}{n}\right) + \sin\left(\frac{2k(t-1)\pi}{n}\right)\right) f_t, \qquad k = 0, 1, \cdots, n-1 \tag{17}$$

7.2 Algorithm for the DCT-IV

For the DCT-IV, pad $p = 3n$ zeros and compute the FFT of the signal $\widehat{f} = G(E_{3n}(\mathrm{f}))$. The DCT-IV coefficients $\mathbf{c} = (c_0, c_2, \ldots, c_{n-1})$, are given by

$$c_k = \sqrt{\frac{2}{n}}\left(\cos\left(\frac{(2k+1)\pi}{4n}\right)\mathrm{Re}(\widehat{\mathrm{f}}_{2\mathrm{k}+1}) + \sin\left(\frac{(2\mathrm{k}+1)\pi}{4\mathrm{n}}\right)\mathrm{Im}(\widehat{\mathrm{f}}_{2\mathrm{k}+1})\right). \tag{18}$$

The inverse DCT-IV is also given by (18), with the coefficients and signal values reversed.

7.3 Boundary Extension Rules

Since the smooth tapers extend beyond the ends of the analysis intervals, it is necessary to extend the data at boundaries. This is similar to the boundary correction algorithms required in wavelet analysis.

For a taper of length $2m \leq n$, it is necessary to extend a signal of length n $(f_1, f_2, \ldots, f_n)$ by m values on each side to a obtain a signal of length $n + 2m$ $(\widetilde{f}_{-m+1}, \widetilde{f}_{-m+2}, \ldots, \widetilde{f}_{n+m})$. There are three boundary extension rules available in S+WAVELETS:

cp.reflect: The signal is reflected at the ends using the same $(+,-)$ polarity as the folding operator. The extended signal is

$$\tilde{f}_i = \begin{cases} f_{-i+1} & \text{for } i = -m+1, -m+2, \cdots, 0 \\ f_i & \text{for } i = 1, 2, \cdots, n \\ f_{2n-i+1} & \text{for } i = n+1, n+2, \cdots, n+m \end{cases}$$

zero: The signal is zero padded, so the extended signal is

$$\tilde{f}_i = \begin{cases} 0 & \text{for } i = -m+1, -m+2, \cdots, 0 \\ f_i & \text{for } i = 1, 2, \cdots, n \\ 0 & \text{for } i = n+1, n+2, \cdots, n+m \end{cases}$$

periodic: The signal is assumed to be periodic and wrapped around at the ends, so the extended signal is

$$\tilde{f}_i = \begin{cases} f_{n-i} & \text{for } i = -m+1, -m+2, \cdots, 0 \\ f_i & \text{for } i = 1, 2, \cdots, n \\ f_{i-n} & \text{for } i = n+1, n+2, \cdots, n+m \end{cases}$$

This is the default boundary extension rule in S+WAVELETS for cosine packet analysis.

Only the periodic boundary extension preserves orthogonality for the boundary blocks. The boundary blocks for the `cp.reflect` and `zero` extensions are only nearly orthogonal.

Note: Instead of extending the data, Meyer [Mey93] suggests modifying the tapers at the boundary. By using a discontinuous "boxcar" taper, there is no need to extend the signal beyond the boundary. This approach leads to an orthogonal transform.

8 Computing the Cosine Packet Transform

Let $X = (x_1, ..., x_n)$ be the signal and partition the signal into p contiguous blocks: $B_1, \ldots, B_p$. The jth block has n_j coefficients and is given by

$$\mathbf{B}_j = (f_{i_j}, \cdots, f_{i_j+n_j-1})$$

where $i_1 = 1$ and $i_{j+1} = i_j + n_j$ for $j = 1, \ldots, p-1$. Let $2m$ be the length of the taper; Each block B_i is of length 2^{J-j}. Let $2m$ be the length of the taper where m is less than or equal to half of the length of the *shortest* block. Let length m vectors

$$\begin{aligned} b_{\text{int}} &= (\beta(\frac{1}{2m}), \beta(\frac{3}{2m}), \ldots, \beta(\frac{2m-1}{2m})) \\ b_{\text{ext}} &= \sqrt{1 - b_{\text{int}}^2} \end{aligned}$$

be the weights of the interior and exterior taper window respectively. See section B below for the tapering functions $\beta(t)$.

The cosine packet transform is computed by applying four basic operators to each block $\mathbf{B}_j$:

1. Use boundary rule to create boundary blocks $\mathbf{B}_0$ and $\mathbf{B}_{p+1}$:

   - **periodic**

   $$\begin{aligned} B_0 &= B_p \\ B_{p+1} &= B_0 \end{aligned}$$

   Note that we only need the first m values.

   - **zero**

   $$B_0 = B_{p+1} = (\overbrace{0, ..., 0}^{m})$$

   - **cp.reflect**

   $$\begin{aligned} B_0 &= \text{rev}(B_1) \\ B_{p+1} &= -\text{rev}(B_p) \end{aligned}$$

   Note that blocks B_0 and B_{p+1} require only m values.

2. Extend block $\mathbf{B}_j$ on the left and right using the neighboring blocks $\mathbf{B}_{j-1}$ and $\mathbf{B}_{j+1}$ to obtain the extended signal $\tilde{\mathbf{f}} = (\tilde{f}_{-m+1}, \tilde{f}_{-m+2}, \ldots, \tilde{f}_{n_j+m})$:

$$\tilde{f}_i = \begin{cases} f_{j-1,n_{j-1}+i} & \text{for } i = -m+1, -m+2, \cdots, 0 \\ f_{j,i} & \text{for } i = 1, 2, \cdots, n_j \\ f_{j+1,i} & \text{for } i = n_j+1, n_j+2, \cdots, n_j+m \end{cases} \tag{19}$$

   For a block of length n_j, the signal $\tilde{\mathbf{f}}$ is of length $2m + n_j$.

3. Apply the interior and exterior taper windows b_{int} and b_{ext} to block $\tilde{\mathbf{f}}$ to obtain the tapered block $\mathbf{g}$ as follows:

$$g_i = \begin{cases} b_{\text{ext}}(-i+1)\tilde{f}_i & \text{for } i = -m+1, -m+2, \cdots, 0 \\ b_{\text{int}}(i)\tilde{f}_i & \text{for } i = 1, 2, \cdots, m \\ \tilde{f}_i & \text{for } i = m+1, m+2, \cdots, n_j - m \\ b_{\text{int}}(n_j - i + 1)\tilde{f}_i & \text{for } i = n_j - m + 1, n_j - m + 2, \cdots, n_j \\ b_{\text{ext}}(i - n_j)\tilde{f}_i & \text{for } i = n_j + 1, n_j + 2, \cdots, n_j + m \end{cases} \tag{20}$$

4. Fold m values on each end as follows:

$$\tilde{g}_i = \begin{cases} g_i + g_{-i+1} & \text{for } i = 1, 2, \cdots, m \\ g_i & \text{for } i = m+1, m+2, \cdots, n_j - m \\ g_i - g_{n_j+(n_j-i)+1} & \text{for } i = n_j - m + 1, n_j - m + 2, \cdots, n_j \end{cases} \tag{21}$$

This yields a "folded" signal $\tilde{\mathbf{g}} = (\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_{n_j})$ of length n_j. This folding is said to have $(+,-)$ polarity, since the reflected signal is added on the left boundary and subtracted on the right boundary.

5. Apply the DCT to the tapered and folded signal $\tilde{\mathbf{g}}$ to obtain the CPT coefficients $\mathbf{c}_j = (c_{j,1}, c_{j,2}, \ldots, c_{j,n_j})$.

9 Computing the Inverse Cosine Packet Transform

The inverse cosine packet transform is obtained by reversing the steps of the forward cosine packet transform. Initialize the output signal $\mathbf{f}$ to a vector of n zeros (0, 0, ..., 0). For each block $j = 1, \ldots, p$, of CPT coefficients $\mathbf{c}_j$, do the following four steps:

1. Apply the inverse DCT to the coefficients $\mathbf{c}_j$ to obtain the tapered and folded signal $\tilde{\mathbf{g}}$.

2. Unfold the signal using the $(+,-)$ polarity to obtain the unfolded signal $\mathbf{g}$ of length $n + 2m$.

$$g_i = \begin{cases} \tilde{g}_{-i+1} & \text{for } i = -m+1, -m+2, \cdots, 0 \\ \tilde{g}_i & \text{for } i = 1, m+2, \cdots, n_j \\ -\tilde{g}_{n_j+(n_j-i+1)} & \text{for } i = n_j+1, n_j+2, \cdots, n_j+m \end{cases} \tag{22}$$

3. Untaper $\mathbf{g}$ to obtain the extended signal $\tilde{\mathbf{f}}$:

$$\tilde{f}_i = \begin{cases} g_i / b_{\text{ext}}(-i+1) & \text{for } i = -m+1, -m+2, \cdots, 0 \\ g_i / b_{\text{int}}(i) & \text{for } i = 1, 2, \cdots, m \\ g_i & \text{for } i = m+1, m+2, \cdots, n_j - m \\ g_i / b_{\text{int}}(n_j - i + 1) & \text{for } i = n_j - m + 1, n_j - m + 2, \cdots, n_j \\ g_i / b_{\text{ext}}(i - n_j) & \text{for } i = n_j + 1, n_j + 2, \cdots, n_j + m \end{cases} \tag{23}$$

4. Separate $\tilde{\mathbf{f}}$ into three blocks: left block $\mathbf{B}_j^{(\ell)}$, central block $\mathbf{B}_j^{(c)}$ and right block $\mathbf{B}_j^{(r)}$:

$$\mathbf{B}_j^{(\ell)} = (0, \cdots, 0, \tilde{f}_{-m+1}, \cdots, \tilde{f}_0)$$

$$\begin{aligned}
\mathbf{B}_j^{(c)} &= (\tilde{f}_1, \cdots, \tilde{f}_{n_j}) \\
\mathbf{B}_j^{(r)} &= (\tilde{f}_{n_j+1}, \cdots, \tilde{f}_{n_j+m}, 0, \cdots, 0)
\end{aligned}$$

Now accumlate the blocks as follows to produce the blocks for the output signal:

$$\begin{array}{cccccccc}
B_1^{(\ell)} & B_1^{(c)} & B_1^{(r)} & & & & & \\
 & B_2^{(\ell)} & B_2^{(c)} & B_2^{(r)} & & & & \\
 & & B_3^{(\ell)} & B_3^{(c)} & B_3^{(r)} & & & \\
 & & & \vdots & \vdots & \vdots & & \\
+ & & & & & B_p^{(\ell)} & B_p^{(c)} & B_p^{(r)} \\
\hline
 & B_1 & B_2 & & & B_{p-1} & B_p &
\end{array}$$

The signal blocks $B_1, ..., B_p$ are constructed by adding the appropriate left, central and right blocks, e.g. $B_2 = B_1^{(r)} + B_2^{(c)} + B_3^{(\ell)}$.

Correct the boundary blocks B_1 and B_p based on the boundary extension rule:

- **periodic:**

$$\begin{aligned}
B_1 &= B_1 + B_p^{(r)} \\
B_p &= B_p + B_1^{(\ell)}.
\end{aligned}$$

- **zero:**

$$\begin{aligned}
(B_1(1), ..., B_1(m)) &= \frac{(\tilde{B}_1(1), ..., \tilde{B}_1(m))}{b_{\text{int}}} \\
(B_p(n), ..., B_p(n-m+1)) &= \frac{(\tilde{B}_p(n), ..., \tilde{B}_p(n-m+1))}{b_{\text{int}}}.
\end{aligned}$$

- **cp.reflect:**

$$\begin{aligned}
(B_1(1), ..., B_1(m)) &= \frac{(B_1(1), ..., B_1(m))}{b_{\text{int}}(b_{\text{int}} + b_{\text{ext}})} \\
(B_p(n), ..., B_p(n-m+1)) &= \frac{(B_p(n), ..., B_p(n-m+1))}{b_{\text{int}}(b_{\text{int}} + b_{\text{ext}})}.
\end{aligned}$$

A Polynomial and Zero Extrapolation

In particular, for $d = 0$, a_i's are repeated values of $Ave(x_1, \ldots, x_Q)$. Note that for $d = -1$, $(a_1, \ldots, a_q) = (b_1, \ldots, b_q) = (0, \ldots, 0)$.

For general polynomial extrapolation, define a Vandermonde matrix of $(t_1, \ldots, t_n)$:

$$V_m(t_1, \ldots, t_n) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ t_1 & t_2 & \ldots & t_n \\ \vdots & \vdots & & \vdots \\ t_1^m & t_2^m & \ldots & t_n^m \end{pmatrix}_{(m+1)\times n}$$

and $V = V_d(0, \frac{1}{Q-1}, \ldots, \frac{Q-2}{Q-1}, 1)$, $V_\ell = V_d(\frac{-q}{Q-1}, \ldots, \frac{-1}{Q-1})$ and $V_r = V_d(\frac{Q}{Q-1}, \ldots, \frac{Q+q-1}{Q-1})$. Then

$$\begin{pmatrix} a_1 \\ \vdots \\ a_q \end{pmatrix} = V_\ell'(VV')^{-1}V \begin{pmatrix} x_1 \\ \vdots \\ x_Q \end{pmatrix}$$

and

$$\begin{pmatrix} b_1 \\ \vdots \\ b_q \end{pmatrix} = V_r'(VV')^{-1}V \begin{pmatrix} x_{n-Q+1} \\ \vdots \\ x_n \end{pmatrix}.$$

B Tapers for Orthogonal Cosine Packets

Very special tapering functions β_I are needed for the cosine packets transforms. Because the cosine packet functions overlap, only certain types of tapers preserve orthogonality in cosine packet analysis (see [AWW92, Wic94]). S+WAVELETS offers 7 different tapers β_I which preserve orthogonality:

`boxcar, poly1, poly2, poly3, poly4, poly5, trig`

The `boxcar` taper is discontinuous, and is used for the block DCT. The `poly1` is a polynomial taper with one continuous derivative. Likewise, the `poly2`, ..., `poly5` are polynomial tapers with 2–5 continuous derivatives. The `trig` taper is based on trigonometric functions, and is the smoothest taper available in S+WAVELETS. The default taper in S+WAVELETS is the `poly2` taper.

A tapering function β_I defined for an interval $I = [\alpha, \beta]$ has the form

$$\beta_I(t) = \begin{cases} 0 & t \le \alpha - \delta_\alpha \\ \mu(\frac{t-\alpha+\delta_\alpha}{2\delta_\alpha}) & \alpha - \delta_\alpha < t < \alpha + \delta_\alpha \\ 1 & \alpha + \delta_\alpha \le t \le \beta - \delta_\beta \\ \mu(\frac{\beta+\delta_\beta-t}{2\delta_\beta}) & \beta - \delta_\beta < t < \beta + \delta_\beta \\ 0 & t \ge \beta + \delta_\beta \end{cases} \tag{24}$$

where $\mu(\cdot)$ is the left-bell function satisfying

$$\mu(t) = \begin{cases} 0 & t \le 0 \\ 1 & t > 1 \end{cases}$$

and $\mu(t) + \mu(1-t) = 1$. The parameter $\delta_\alpha = \epsilon_\alpha \Delta_I$, with $\Delta_I = \beta - \alpha$, defines the tapering region for the left hand side of the interval; and $\delta_\beta = \epsilon_\beta \Delta_I$ defines the tapering region for the right hand side, with the constraint $\epsilon_\alpha + \epsilon_\beta \leq 1$.

The specific form of these tapers is as follows:

boxcar The boxcar tapering function is given by

$$\mu(t) = \begin{cases} 0 & \text{if } t \leq 1/2 \\ 1 & \text{if } t > 1/2 \end{cases}$$

poly1 , ..., poly5 A polynomial tapering function of order $p = 1, \ldots, 5$ is given by

$$\mu_p(t) = \begin{cases} 0 & t < 0 \\ \sqrt{t^{p+1} \sum_{k=0}^{p} b_k t^k} & 0 \leq t \leq 1 \\ 1 & t > 1 \end{cases}$$

where $(b_0, b_1, \ldots, b_p)' = A_p^{-1}(1, 0, \ldots, 0)'$ with

$$A_p = (a_{ij})_{0 \leq i,j \leq p} \qquad a_{ij} = \binom{p+j-1}{i}.$$

The polynomial tapers are available for $p = 1, 2, \ldots, 5$, with higher order tapers providing a greater degree of smoothness.

trig The tapering function for the trigonometric window is given by

$$\mu(t) = \begin{cases} 0 & t < 0 \\ \sin\left(\frac{\pi}{4}\left(1 - \cos(\pi t)\right)\right) & 0 \leq t \leq 1 \\ 1 & t > 1 \end{cases}$$

The trigonometric taper is the smoothest taper available in S+WAVELETS.

References

[AWW92] P. Auscher, G. Weiss, and M. V. Wickerhauser. Local sine and cosine bases of Coifman and Meyer and the construction of smooth wavelets. In Charles K. Chui, editor, *Wavelets: a tutorial in theory and applications*, pages 237–256. Academic Press, Inc., San Diego, CA, 1992.

[BA83] P. J. Burt and E. H. Adelson. The laplacian pyramid transforms for image coding. *IEEE Transactions on Communications*, 31:532–540, 1983.

[BG94] Andrew G. Bruce and Hong-Ye Gao. *S+WAVELETS Users Manual.* StatSci Division of MathSoft, Inc., 1700 Westlake Ave. N, Seattle, WA 98109-9891, 1994.

[BG95] Andrew G. Bruce and Hong-Ye Gao. S+WAVELETS: An object-oriented toolkit for wavelet analysis. Technical report, StatSci Division of MathSoft, Inc., 1700 Westlake Ave. N, Seattle, WA 98109-9891, 1995.

[Bri92] Christopher M. Brislawn. Classification of symmetric wavelet transforms. Technical report, Los Alamos National Laboratory, Los Alamos, New Mexico, 87545, 1992.

[CDV93] A. Cohen, I. Daubechies, and P. Vial. Wavelets on the interval and fast wavelet transforms. *Applied and Computational Harmonic Analysis*, 1:54–81, 1993.

[Chu92] C. K. Chui. *An introduction to wavelets.* Academic Press, Inc., San Diego, CA, 1992.

[CM91] R. Coifman and Y. Meyer. Remarques sur l'analyse de Fourier à fenêtre. *C. R. Acad. Sci. Paris*, 312:259–261, 1991.

[CMQW90] R. Coifman, Y. Meyer, S. Quake, and V. Wickerhauser. Signal processing and compression with wavelet packets. Technical report, Yale University, 1990.

[CMW92] R. Coifman, Y. Meyer, and V. Wickerhauser. Wavelet analysis and signal processing. In *Wavelets and Their Applications*, pages 153–178. Jones and Bartlett Publishers, Boston, 1992.

[CW92] R. Coifman and V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38(2):713–718, 1992.

[Dau92] I. Daubechies. *Ten lectures on wavelets.* Society for industrial and applied mathematics, Philadelphia, PA, 1992.

[Dut87] P. Dutilleux. An implementation of the "algorithme à trous" to compute the wavelet transform. In J. M. Combes, A. Grossman, and Ph. Tchamitchian, editors, *Wavelets: Time-Frequency Methods and Phase Space*, pages 298–304. Springer-Verlag, 1987.

[JLS94] Bjorn Jawerth, Yi Liu, and Wim Sweldens. New folding operators for image compression. In *SPIE Proceedings, Wavelet Applications*, volume 2242, Orlando, FL, April 1994.

[Mal86] H. S. Malvar. Fast computation of the discrete cosine transform through the fast Hartley transform. *Electronic Letters*, 22(7):352–353, March 1986.

[Mal89] Stéphane Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[Mey93] Yves Meyer. *Wavelets: Algorithms and Applications.* SIAM, 3600 University City Science Center, Philadelphia, PA 19104-2688, 1993.

[RY90] K. R. Rao and P. Yip. *Discrete Cosine Transform.* Academic Press, Inc., San Diego, CA, 1990.

[She92] Mark J. Shensa. The discrete wavelet transform: Wedding the À trous and Mallat algorithms. *IEEE Transactions on Signal Processing*, 40(10):2464–2482, 1992.

[SJBH85] H. V. Sorenson, D. L. Jones, C. S. Burrus, and M. T. Heideman. On computing the discrete Hartley transfrom. *IEEE Trans. ASSP*, 33:1231–1245, 1985.

[SMB94] Jean-Luc Starck, Fionn Murtagh, and Albert Bijanoui. Multiresolution support applied to image filtering and restoration. Technical report, Observatoire de la Côte d'Azur, B. P. 229, F-06304 Nice Cedex 4, France, 1994.

[Str89] Gilbert Strang. Wavelets and dilation equations: A brief introduction. *SIAM Review*, 31(4):614–627, 1989.

[Wic94] Mladen V. Wickerhauser. *Adapted Wavelet Analysis – from theory to software.* A. K. Peters, Ltd, 1994.

OFFICE OF THE UNDER SECRETARY OF DEFENSE (ACQUISITION)
DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22304-6145

IN REPLY REFER TO DTIC-OCC

SUBJECT: Distribution Statements on Technical Documents

TO: OFFICE OF NAVAL RESEARCH
CORPORATE PROGRAMS DIVISION
ONR 353
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

1995 1024 092

1. Reference: DoD Directive 5230.24, Distribution Statements on Technical Documents, 18 Mar 87.

2. The Defense Technical Information Center received the enclosed report (referenced below) which is not marked in accordance with the above reference.

TECHNICAL DETAILS REPORT
N00014-92-0066
TITLE: S+ WAVELENGTHS:
ALGORITHMS AND TECHNICAL
DETAILS

3. We request the appropriate distribution statement be assigned and the report returned to DTIC within 5 working days.

4. Approved distribution statements are listed on the reverse of this letter. If you have any questions regarding these statements, call DTIC's Cataloging Branch, (703) 274-6837.

FOR THE ADMINISTRATOR:

1 Encl

GOPALAKRISHNAN NAIR
Chief, Cataloging Branch

FL-171
Jul 93

DISTRIBUTION STATEMENT A:

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

DISTRIBUTION STATEMENT B:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES ONLY; (Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT C:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND THEIR CONTRACTORS; (Indicate Reason and Date Below). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT D:

DISTRIBUTION AUTHORIZED TO DOD AND U.S. DOD CONTRACTORS ONLY; (Indicate Reason and Date Below). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT E:

DISTRIBUTION AUTHORIZED TO DOD COMPONENTS ONLY; (Indicate Reason and Date Below). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office Below).

DISTRIBUTION STATEMENT F:

FURTHER DISSEMINATION ONLY AS DIRECTED BY (Indicate Controlling DoD Office and Date Below) or HIGHER DOD AUTHORITY.

DISTRIBUTION STATEMENT X:

DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND PRIVATE INDIVIDUALS OR ENTERPRISES ELIGIBLE TO OBTAIN EXPORT-CONTROLLED TECHNICAL DATA IN ACCORDANCE WITH DOD DIRECTIVE 5230.25, WITHHOLDING OF UNCLASSIFIED TECHNICAL DATA FROM PUBLIC DISCLOSURE, 6 Nov 1984 (Indicate date of determination). CONTROLLING DOD OFFICE IS (Indicate Controlling DoD Office).

The cited documents has been reviewed by competent authority and the following distribution statement is hereby authorized.

A

(Statement)

OFFICE OF NAVAL RESEARCH
CORPORATE PROGRAMS DIVISION
ONR 353
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

(Controlling DoD Office Name)

(Reason)

(Controlling DoD Office Address, City, State, Zip)

DEBRA T. HUGHES
DEPUTY DIRECTOR
CORPORATE PROGRAMS OFFICE

(Signature & Typed Name)

(Assigning Office)

19 SEP 1995

(Date Statement Assigned)